SPARK2014

Formal Program Verification For All

High Confidence Software and Systems Conference, Annapolis, May 2014

Yannick Moy, AdaCore, moy@adacore.com

AdaCore ALTRAN

PARTNERSHIP

SPARK is not Ada

SPARK 2014 is Ada 2012

SPARK 2014 is
contract-based practical static verification for Ada

SPARK 2014 is
contract-based practical static verification for Ada

modular

scalable

user-defined properties

precise

SPARK 2014 is
contract-based practical static verification for Ada

executable contracts

integration in IDEs

proof automation

SPARK 2014 is
contract-based practical static verification for Ada

dataflow analysis

information flow analysis

robustness analysis

functional analysis

SPARK 2014 is
contract-based practical static verification for Ada

usable on existing codebase

large subset

combined with testing

File Edit Navigate VCS Project Build Debug Tools CodePeer SPARK Window Help

search

Project Outline

alarm.ads admintoken.ads admintoken.adb admin.ads

Tokeneer

- admin.adb
- admin.ads
- admintoken-interfac.adb
- admintoken-interfac.ads
- admintoken.adb
- admintoken.ads
- alarm-interfac.adb
- alarm-interfac.ads
- alarm.adb
- alarm.ads
- alarmapi.adb
- alarmapi.ads
- alarmtypes.ads
- auditlog.adb
- auditlog.ads
- audittypes.ads
- basictypes.ads
- bio-interfac.adb
- bio-interfac.ads
- bio.adb
- bio.ads
- bioapi.adb
- bioapi.ads
- cert-attr-auth.adb
- cert-attr-auth.ads
- cert-attr-ianda.adb
- cert-attr-ianda.ads
- cert-attr-priv.adb
- cert-attr-priv.ads
- cert-attr.adb
- cert-attr.ads
- cert-id.adb
- cert-id.ads
- cert.adb
- cert.ads
- certificatestore.adb
- certificatestore.ads
- certproc.adb
- certproc.ads
- certprocessing.adb

```
1  with Door; use Door;
2  with AlarmTypes; use AlarmTypes;
3  --------------------------------------------------------------
4  -- Tokeneer ID Station Core Software
5  --
6  -- Copyright (2003) United States Government, as represented
7  -- by the Director, National Security Agency.All rights reserved.
8  --
9  -- This material was originally developed by Praxis High Integrity
10 -- Systems Ltd.under contract to the National Security Agency.
11 --------------------------------------
12
13 --------------------------------------------------------------
14 -- Alarm
15 --
16 -- Description:
17 --    Provides interface to the alarm
18 --
19 --------------------------------------------------------------
20
21 with AuditLog;
22
23 package Alarm is
24
25    ------------------------------------------------------------
26    -- PROOF A   ATIONS FOR S   TY PROPE   3                   --
27    ------------------------------------------------------------
28    -- A proof   ction is requir   mode   e proof              --
29    -- functio   terfac.prf_isAla   g(Int   c.Output)          --
30    -- (which    ot visible outsi   he pa                      --
31    -- Interfa   tput is a refine   of O   t, so               --
32    -- need to   e Output as a pa   ter o   e                  --
33    -- functio   do this, need   efine   bstract              --
34    -- type fo   tput.                                        --
35    -- The Int   proof fu   n is                              --
36    -- effecti   t of this pro                                --
37    ------------------------------------------------------------
38    function IsAlarming return Boolean
39      with Global     => null,
40           Convention => Ghost;
41
42    --------------------------------------------------------------
43    -- UpdateDevice
44    --
45    -- Description:
46    --    Updates the physical alarm depending on the state of the
47    --    Door alarm and the AuditLog alarm.
48    --
```

11:35

Messages Locations

```
gnatprove -P/Users/moy/tokeneer/tokeneer.gpr --clean

[2014-04-25 15:14:33] process terminated successfully, elapsed time: 00.20s
```

DEMO

File Edit Navigate VCS Project Build Debug Tools CodePeer SPARK Window Help

search

Project Outline

alarm.ads admintoken.ads admintoken.adb admin.ads

Tokeneer

- admin.adb
- admin.ads
- admintoken-interfac.adb
- admintoken-interfac.ads
- admintoken.adb
- admintoken.ads
- alarm-interfac.adb
- alarm-interfac.ads
- alarm.adb
- alarm.ads
- alarmapi.adb
- alarmapi.ads
- alarmtypes.ads
- auditlog.adb
- auditlog.ads
- audittypes.ads
- basictypes.ads
- bio-interfac.adb
- bio-interfac.ads
- bio.adb
- bio.ads
- bioapi.adb
- bioapi.ads
- cert-attr-auth.adb
- cert-attr-auth.ads
- cert-attr-ianda.adb
- cert-attr-ianda.ads
- cert-attr-priv.adb
- cert-attr-priv.ads
- cert-attr.adb
- cert-attr.ads
- cert-id.adb
- cert-id.ads
- cert.adb
- cert.ads
- certificatestore.adb
- certificatestore.ads
- certproc.adb
- certproc.ads
- certprocessing.adb

```
1  with Door; use Door;
2  with AlarmTypes; use AlarmTypes;
3  ------------------------------------------------------------------
4  -- Tokeneer ID Station Core Software
5  --
6  -- Copyright (2003) United States Government, as represented
7  -- by the Director, National Security Agency.All rights reserved.
8  --
9  -- This material was originally developed by Praxis High Integrity
10 -- Systems Ltd.under contract to the National Security Agency.
11 ------------------------------------|-----------------------------
12
13 ------------------------------------------------------------------
14 -- Alarm
15 --
16 -- Description:
17 --    Provides interface to the alarm
18 --
19 ------------------------------------------------------------------
20
21 with AuditLog;
22
23 package Alarm is
24
25    ------------------------------------------------------
26    -- PROOF ANNOTATIONS FOR SECURITY PROPERTY 3         --
27    --====================================================--
28    -- A proof function is required to model the proof   --
29    -- function Interfac.prf_isAlarming(Interfac.Output) --
30    -- (which is not visible outside the package body).  --
31    -- Interfac.Output is a refinement of Output, so     --
32    -- need to take Output as a parameter of the         --
33    -- function.To do this, need to define an abstract   --
34    -- type for Output.                                  --
35    -- The Interfac.prf_isAlarming proof function is     --
36    -- effectively a refinement of this proof function.  --
37    ------------------------------------------------------
38    function IsAlarming return Boolean
39      with Global     => null,
40           Convention => Ghost;
41
42    ------------------------------------------------------------------
43    -- UpdateDevice
44    --
45    -- Description:
46    --    Updates the physical alarm depending on the state of the
47    --    Door alarm and the AuditLog alarm.
48    --
```

11:35

Messages Locations

```
gnatprove -P/Users/moy/tokeneer/tokeneer.gpr --clean

[2014-04-25 15:14:33] process terminated successfully, elapsed time: 00.20s
```

Limitations of Vintage SPARK

1. **cost of adding mandatory contracts**
2. **not usable on existing code**
   - constraints on visibility / program structure
   - very restricted language subset
   - constraints on the control flow graph
3. **limitations of proof**
   - floating-point interpreted as real
   - very simple VC generation
   - prover does not handle well disjunctions and quantifiers
4. **not integrated in traditional development process**
   - incompatible with testing
   - impossible to debug contracts
   - weak IDE support

Strengths of SPARK 2014

1. cost of adding mandatory contracts
2. not usable on existing code
   - constraints on visibility / program structure
   - very restricted language subset
   - constraints on the control flow graph
3. limitations of proof
   - floating-point interpreted as real
   - very simple VC generation
   - prover does not handle well disjunctions and quantifiers
4. not integrated in traditional development process
   - incompatible with testing
   - impossible to debug contracts
   - weak IDE support

Strengths of SPARK 2014

1. generation of required contracts
2. not usable on existing code
   - constraints on visibility / program structure
   - very restricted language subset
   - constraints on the control flow graph
3. limitations of proof
   - floating-point interpreted as real
   - very simple VC generation
   - prover does not handle well disjunctions and quantifiers
4. not integrated in traditional development process
   - incompatible with testing
   - impossible to debug contracts
   - weak IDE support

Strengths of SPARK 2014

1. generation of required contracts
2. usable on existing code
   - use Ada rules for visibility / program structure
   - subset includes generics, discriminants, etc.
   - allow any loop exit, early return, recursion
3. limitations of proof
   - floating-point interpreted as real
   - very simple VC generation
   - prover does not handle well disjunctions and quantifiers
4. not integrated in traditional development process
   - incompatible with testing
   - impossible to debug contracts
   - weak IDE support

Strengths of SPARK 2014

1. generation of required contracts
2. usable on existing code
   - use Ada rules for visibility / program structure
   - subset includes generics, discriminants, etc.
   - allow any loop exit, early return, recursion
3. powerful automatic proof
   - support IEEE 754 floating-point semantics
   - efficient and precise VC generation
   - use state-of-the-art SMT solver
4. not integrated in traditional development process
   - incompatible with testing
   - impossible to debug contracts
   - weak IDE support

Strengths of SPARK 2014

1. generation of required contracts
2. usable on existing code
   - use Ada rules for visibility / program structure
   - subset includes generics, discriminants, etc.
   - allow any loop exit, early return, recursion
3. powerful automatic proof
   - support IEEE 754 floating-point semantics
   - efficient and precise VC generation
   - use state-of-the-art SMT solver
4. integrated in developer toolbox
   - combined with testing
   - contracts can be executed and debugged
   - fine-grain interactions in two IDEs

Essential Principles of the Retooling

1. **convergence with compiler technology (GNAT)**
   - allows to support a larger subset of Ada in SPARK
   - target-dependent & compiler-dependent proofs

2. **use of intermediate verification language (Why3)**
   - powerful VC generation and transformations
   - rich language features (exceptions, types)

3. **use of state-of-the-art SMT solvers (Alt-Ergo + ...)**
   - powerful automation of proofs

Tool Architecture

note: all components are Free / Libre / Open Source Software

GNAT project support

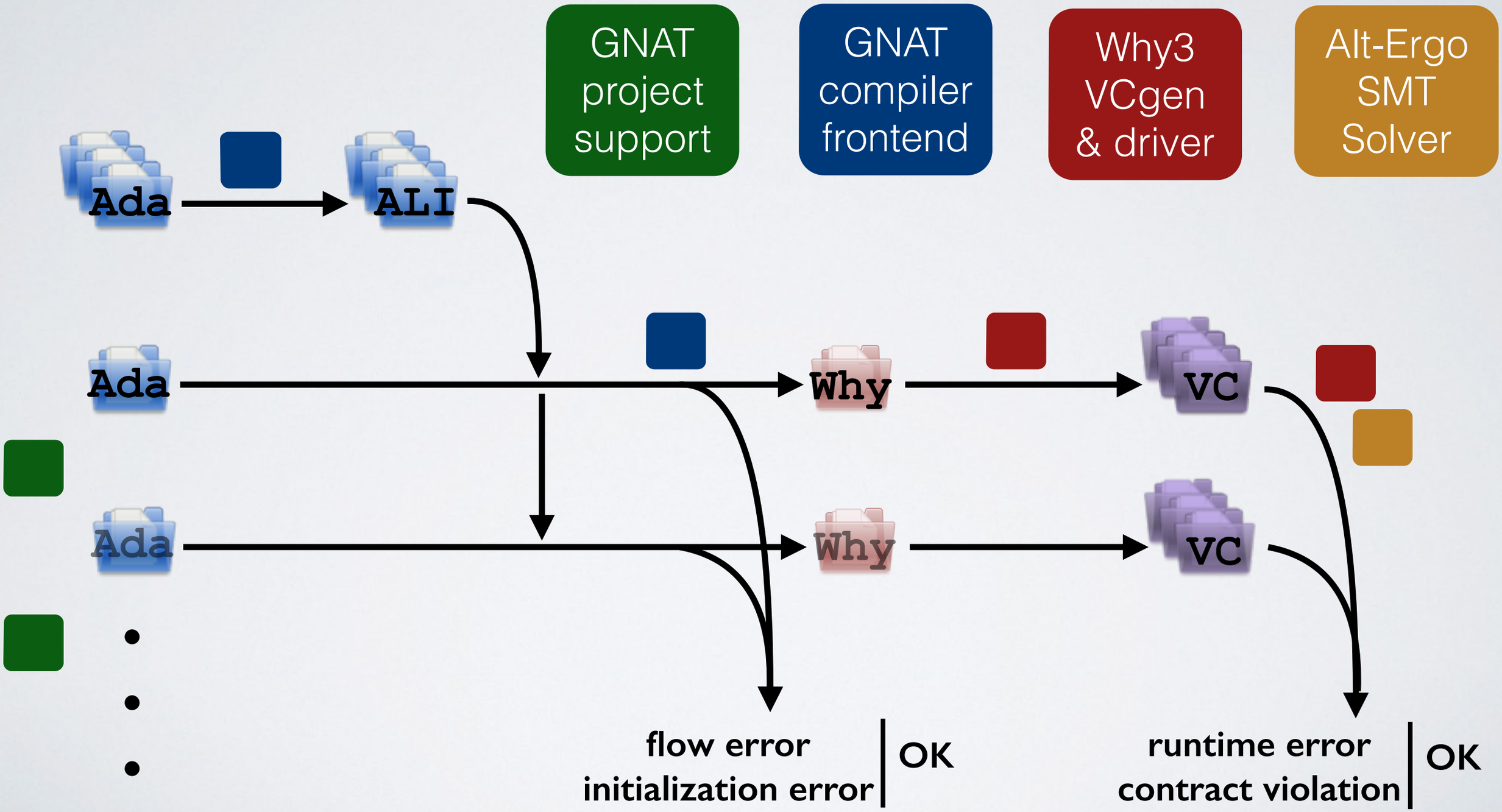
GNAT compiler frontend

Why3 VCgen & driver

Alt-Ergo SMT Solver

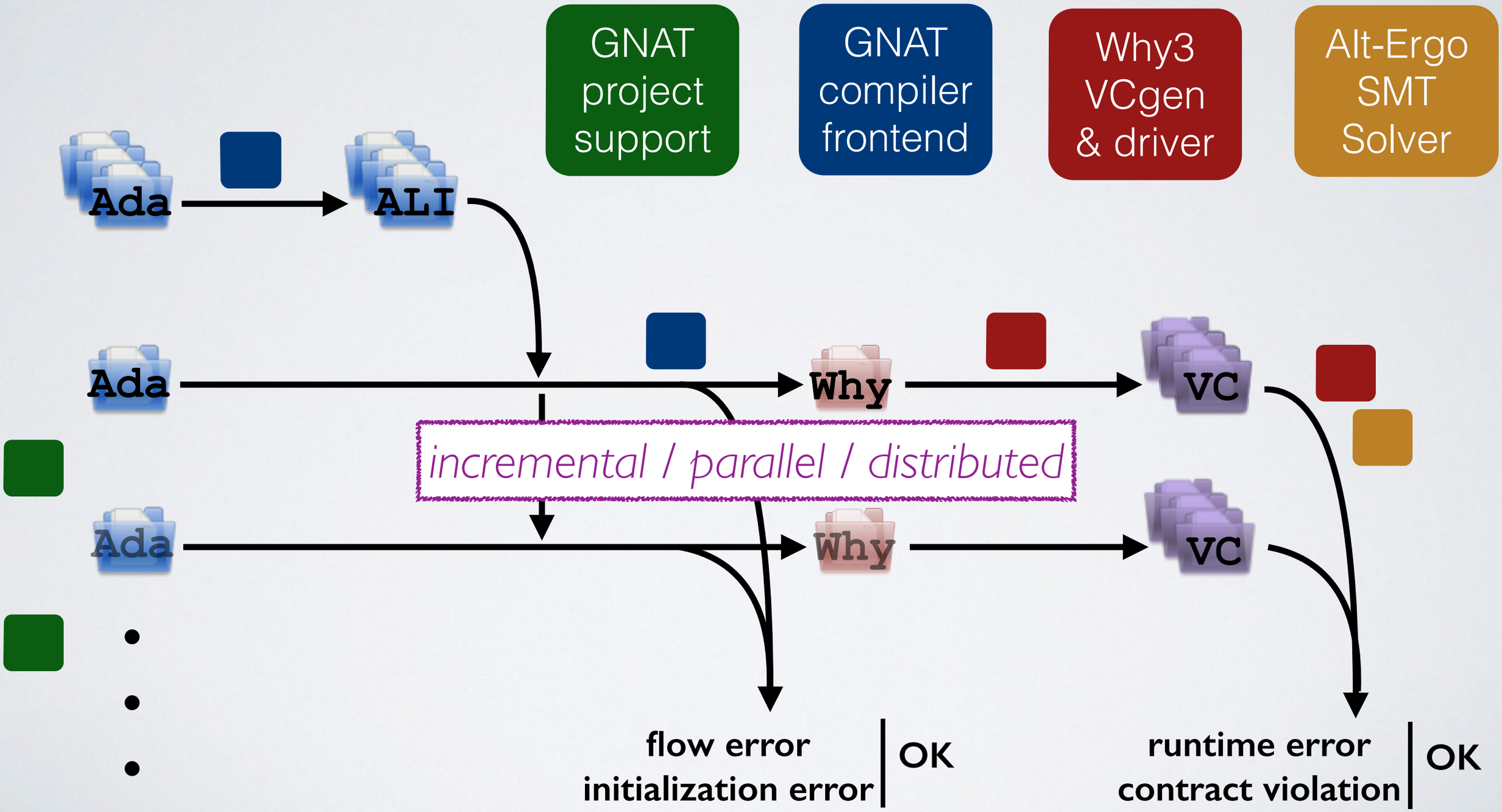
Tool Architecture

note: all components are Free / Libre / Open Source Software



Tool Architecture

note: all components are Free / Libre / Open Source Software



Case Studies by Airbus Defence & Space

carried by David Lesens, expert in formal methods
in *Formal Validation of Aerospace Software*, DASIA 2013

with Vintage SPARK

(2011, 182 subp, 44 Pre, 66 Post)

with SPARK 2014

(2010-2013, 1500 subp, 1400 Pre, 400 Post)

Case Studies by Airbus Defence & Space

carried by David Lesens, expert in formal methods
in *Formal Validation of Aerospace Software*, DASIA 2013

with Vintage SPARK

(2011, 182 subp, 44 Pre, 66 Post)

+ proved absence of RE

with SPARK 2014

(2010-2013, 1500 subp, 1400 Pre, 400 Post)

+ proved absence of RE (93%)
+ proved functional behavior (98%)

Case Studies by Airbus Defence & Space

carried by David Lesens, expert in formal methods
in *Formal Validation of Aerospace Software*, DASIA 2013

with Vintage SPARK

(2011, 182 subp, 44 Pre, 66 Post)

\+ proved absence of RE

− language restrictions cause over-cost & limited scope

with SPARK 2014

(2010-2013, 1500 subp, 1400 Pre, 400 Post)

\+ proved absence of RE (93%)

\+ proved functional behavior (98%)

\+ generics and discriminants support variability in versions & data

Case Studies by Airbus Defence & Space

carried by David Lesens, expert in formal methods
in *Formal Validation of Aerospace Software*, DASIA 2013

with Vintage SPARK

(2011, 182 subp, 44 Pre, 66 Post)

\+ proved absence of RE

− language restrictions cause over-cost & limited scope

− unacceptable to engineers

with SPARK 2014

(2010-2013, 1500 subp, 1400 Pre, 400 Post)

\+ proved absence of RE (93%)

\+ proved functional behavior (98%)

\+ generics and discriminants support variability in versions & data

\+ usable by non experts

Case Studies by Airbus Defence & Space

carried by David Lesens, expert in formal methods
in *Formal Validation of Aerospace Software*, DASIA 2013

with Vintage SPARK

(2011, 182 subp, 44 Pre, 66 Post)

\+ proved absence of RE

− language restrictions cause over-cost & limited scope

− unacceptable to engineers

− not combined with test

with SPARK 2014

(2010-2013, 1500 subp, 1400 Pre, 400 Post)

\+ proved absence of RE (93%)

\+ proved functional behavior (98%)

\+ generics and discriminants support variability in versions & data

\+ usable by non experts

\+ contracts used for test and proof

Case Studies by Airbus Defence & Space

carried by David Lesens, expert in formal methods
in *Formal Validation of Aerospace Software*, DASIA 2013

with Vintage SPARK

(2011, 182 subp, 44 Pre, 66 Post)

\+ proved absence of RE

− language restrictions cause over-cost & limited scope

− unacceptable to engineers

− not combined with test

− interactive proof required, too complex and expensive

with SPARK 2014

(2010-2013, 1500 subp, 1400 Pre, 400 Post)

\+ proved absence of RE (93%)

\+ proved functional behavior (98%)

\+ generics and discriminants support variability in versions & data

\+ usable by non experts

\+ contracts used for test and proof

\+ use test when not proved

```
-- Set the value of a variable
procedure Set_Nat32_Variable (Variable_Id :        T_Variable_Id;
                              New_Value   :        T_Nat32;
                              Variables   : in out T_Variables)
with
  Pre  =>
    (Is_Valid (Variables) and then
     Is_Nat32 (Variable_Id, Variables) and then
     Get_Min_Nat32 (Variable_Id, Variables) <= New_Value and then
     New_Value <= Get_Max_Nat32 (Variable_Id, Variables)),
  Post =>
    (Is_Valid (Variables) and then
     Is_Nat32 (Variable_Id => Variable_Id,
               Variables   => Variables) and then
     Get_Min_Nat32 (Variable_Id => Variable_Id,
                    Variables   => Variables) = Get_Min_Nat32 (Variable_Id => Variable_Id,
                                                               Variables   => Variables'Old) and then
     Get_Max_Nat32 (Variable_Id => Variable_Id,
                    Variables   => Variables) = Get_Max_Nat32 (Variable_Id => Variable_Id,
                                                               Variables   => Variables'Old) and then
     Get_Nat32 (Variable_Id, Variables) = New_Value and then
       (for all Id in T_Variable_Id =>
          (if Id /= Variable_Id then
           Get_Variable (Id, Variables) = Get_Variable (Id, Variables'Old))));
```

```
-- Set the value of a variable
procedure Set_Nat32_Variable (Variable_Id :        T_Variable_Id;
                              New_Value   :        T_Nat32;
                              Variables   : in out T_Variables)
with
  Pre  =>
    (Is_Valid (Variables) and then
     Is_Nat32 (Variable_Id, Variables) and then
     Get_Min_Nat32 (Variable_Id, Variables) <= New_Value and then
     New_Value <= Get_Max_Nat32 (Variable_Id, Variables)),
  Post =>
    (Is_Valid (Variables) and then
Is_Nat32 (Variable_Id => Variable_Id,
          Variables   => Variables) and then
Get_Min_Nat32 (Variable_Id => Variable_Id,
               Variables   => Variables) = Get_Min_Nat32 (Variable_Id => Variable_Id,
                                                          Variables   => Variables'Old) and then
Get_Max_Nat32 (Variable_Id => Variable_Id,
               Variables   => Variables) = Get_Max_Nat32 (Variable_Id => Variable_Id,
                                                          Variables   => Variables'Old) and then
Get_Nat32 (Variable_Id, Variables) = New_Value and then
  (for all Id in T_Variable_Id =>
     (if Id /= Variable_Id then
      Get_Variable (Id, Variables) = Get_Variable (Id, Variables'Old))));
```

```
-- Set the value of a variable
procedure Set_Nat32_Variable (Variable_Id :        T_Variable_Id;
                              New_Value   :        T_Nat32;
                              Variables   : in out T_Variables)
with
  Pre  =>
    (Is_Valid (Variables) and then
     Is_Nat32 (Variable_Id, Variables) and then
     Get_Min_Nat32 (Variable_Id, Variables) <= New_Value and then
     New_Value <= Get_Max_Nat32 (Variable_Id, Variables)),
  Post =>
    (Is_Valid (Variables) and then
Is_Nat32 (Variable_Id => Variable_Id,
          Variables   => Variables)
Get_Min_Nat32 (Variable_Id => Varia
               Variables   => Varia                                  ) and then

Get_Max_Nat32 (Variable_Id => Vari
               Variables   => V ables) = Get_Max_Nat32 (Variable_Id => Variable_Id,
                                                        Variables   => Variables'Old) and then
Get_Nat32 (Variable_Id, Var ables) = New_Value and then
  (for all Id in T_Variable_Id =>
     (if Id /= Variable_Id then
      Get_Variable (Id, Variables) = Get_Variable (Id, Variables'Old))));
```

rich expressions: quantified (for all, for some), conditionals (if, case)

proof of absence of RE in contracts
(Pre should be self-guarded)

```
-- Set the value of a variable
procedure Set_Nat32_Variable (Variable_Id :         T_Variable_Id;
                              New_Value   :         T_Nat32;
                              Variables   : in out T_Variables)
with
  Pre  =>
    (Is_Valid (Variables) and then
     Is_Nat32 (Variable_Id, Variables) and then
     Get_Min_Nat32 (Variable_Id, Variables) <= New_Value and then
     New_Value <= Get_Max_Nat32 (Variable_Id, Variables)),
  Post =>
    (Is_Valid (Variables) and then
Is_Nat32 (Variable_Id => Variable_Id,
          Variables   => Variables) and then
Get_Min_Nat32 (Variable_Id => Variable_Id,
               Variables   => Variables) = Get_Min_Nat32 (Variable_Id => Variable_Id,
                                                          Variables   => Variables'Old) and then
Get_Max_Nat32 (Variable_Id => Variable_Id,
               Variables   => Variables) = Get_Max_Nat32 (Variable_Id => Variable_Id,
                                                          Variables   => Variables'Old) and then
Get_Nat32 (Variable_Id, Variables) = New_Value and then
  (for all Id in T_Variable_Id =>
     (if Id /= Variable_Id then
      Get_Variable (Id, Variables) = Get_Variable (Id, Variables'Old))));
```

```
-- Set the value of a variable
procedure Set_Nat32_Variable (Variable_Id :        T_Variable_Id;
                              New_Value   :        T_Nat32;
                              Variables   : in out T_Variables)
with
  Pre  =>
    (Is_Valid (Variables) and then
     Is_Nat32 (Variable_Id, Variables) and then
     Get_Min_Nat32 (Variable_Id, Variables) <= New_Value and then
     New_Value <= Get_Max_Nat32 (Variable_Id, Variables)),
  Post =>
    (Is_Valid (Variables) and then
Is_Nat32 (Variable_Id => Variable_Id,
          Variables   => Variables) and then
Get_Min_Nat32 (Variable_Id => Variable_Id,
               Variables   => Variables) = Get_Min_Nat32 (Variable_Id => Variable_Id,
                                                          Variables   => Variables'Old) and then
Get_Max_Nat32 (Variable_Id => Variable_Id,
               Variables   => Variables) = Get_Max_Nat32 (Variable_Id => Variable_Id,
                                                          Variables   => Variables'Old) and then
Get_Nat32 (Variable_Id, Variables) = New_Value and then
  (for all Id in T_Variable_Id =>
     (if Id /= Variable_Id then
      Get_Variable (Id, Variables) = Get_Variable (Id, Variables'Old))));
```

need to express contracts by cases? use Contract_Cases

```
-- Set the value of a variable
procedure Set_Nat32_Variable (Variable_Id :        T_Variable_Id;
                              New_Value   :        T_Nat32;
                              Variables   : in out T_Variables)
with
  Pre  =>
    (Is_Valid (Variables) and then
     Is_Nat32 (Variable_Id, Variables) and then
     Get_Min_Nat32 (Variable_Id, Variables) <= New_Value and then
     New_Value <= Get_Max_Nat32 (Variable_Id, Variables)),
  Post =>
    (Is_Valid (Variables) and then
Is_Nat32 (Variable_Id => Variable_Id,
          Variables   => Variables) and then
Get_Min_Nat32 (Variable_Id => Variable_Id,
               Variables   => Variables) = Get_Min_Nat32 (Variable_Id => Variable_Id,
                                                          Variables   => Variables'Old) and then
Get_Max_Nat32 (Variable_Id => Variable_Id,
               Variables   => Variables) = Get_Max_Nat32 (Variable_Id => Variable_Id,
                                                          Variables   => Variables'Old) and then
Get_Nat32 (Variable_Id, Variables) = New_Value and then
  (for all Id in T_Variable_Id =>
     (if Id /= Variable_Id then
      Get_Variable (Id, Variables) = Get_Variable (Id, Variables'Old))));
```

Expr'Old restricted to minimize surprises to users

```
-- Set the value of a variable
procedure Set_Nat32_Variable (Variable_Id :        T_Variable_Id;
                              New_Value   :        T_Nat32;
                              Variables   : in out T_Variables)
with
  Pre  =>
    (Is_Valid (Variables) and then
     Is_Nat32 (Variable_Id, Variables) and then
     Get_Min_Nat32 (Variable_Id, Variables) <= New_Value and then
     New_Value <= Get_Max_Nat32 (Variable_Id, Variables)),
  Post =>
    (Is_Valid (Variables) and then
     Is_Nat32 (Variable_Id => Variable_Id,
               Variables   => Variables) and then
     Get_Min_Nat32 (Variable_Id => Variable_Id,
                    Variables   => Variables) = Get_Min_Nat32 (Variable_Id => Variable_Id,
                                                               Variables   => Variables'Old) and then
     Get_Max_Nat32 (Variable_Id => Variable_Id,
                    Variables   => Variables) = Get_Max_Nat32 (Variable_Id => Variable_Id,
                                                               Variables   => Variables'Old) and then
     Get_Nat32 (Variable_Id, Variables) = New_Value and then
       (for all Id in T_Variable_Id =>
          (if Id /= Variable_Id then
           Get_Variable (Id, Variables) = Get_Variable (Id, Variables'Old))));
```

need unbounded arithmetic in contract? use Overflow_Mode

EXAMPLE OF LOOP INVARIANT

```
function Init return T_Variables is
   Result : T_Variables;
begin
   pragma Assert (Variable.Is_Valid (Var => Variable.C_Variable));
   for Variable_Id in T_Variable_Id loop
      Result (Variable_Id) := Variable.C_Variable;
      pragma Loop_Invariant
        (for all I in T_Variable_Id range Variable_Id'First .. Variable_Id =>
           Variable.Is_Valid (Result (I)));
   end loop;
   return Result;
end Init;
```

```
function Init return T_Variables is
   Result : T_Variables;
begin
   pragma Assert (Variable.Is_Valid (Var => Variable.C_Variable));
   for Variable_Id in T_Variable_Id loop
      Result (Variable_Id) := Variable.C_Variable;
      pragma Loop_Invariant
        (for all I in T_Variable_Id range T_Variable_Id'First .. Variable_Id =>
           Variable.Is_Valid (Result (I)));
   end loop;
   return Result;
end Init;
```

at run time: like an assertion
in proof: loop invariant (but not Hoare-like)

```
function Init return T_Variables is
   Result : T_Variables;
begin
   pragma Assert (Variable.Is_Valid (Var => Variable.C_Variable));
   for Variable_Id in T_Variable_Id loop
      Result (Variable_Id) := Variable.C_Variable;
      pragma Loop_Invariant
        (for all I in T_Variable_Id range T_Variable_Id'First .. Variable_Id =>
           Variable.Is_Valid (Result (I)));
   end loop;
   return Result;
end Init;
```

```
function Init return T_Variables is
   Result : T_Variables;
begin
   pragma Assert (Variable.Is_Valid (Var => Variable.C_Variable));
   for Variable_Id in T_Variable_Id loop
      Result (Variable_Id) := Variable.C_Variable;
      pragma Loop_Invariant
        (for all I in T_Variable_Id range T_Variable_Id'First .. Variable_Id =>
           Variable.Is_Valid (Result (I)));
   end loop;
   return Result;
end Init;
```

need to refer to value at loop entry? use X'Loop_Entry

```
function Init return T_Variables is
   Result : T_Variables;
begin
   pragma Assert (Variable.Is_Valid (Var => Variable.C_Variable));
   for Variable_Id in T_Variable_Id loop
      Result (Variable_Id) := Variable.C_Variable;
      pragma Loop_Invariant
        (for all I in T_Variable_Id range T_Variable_Id'First .. Variable_Id =>
           Variable.Is_Valid (Result (I)));
   end loop;
   return Result;
end Init;
```

now: methodology for writing loop invariants
soon: common patterns of loop invariants
planned: generation of loop invariants based on patterns

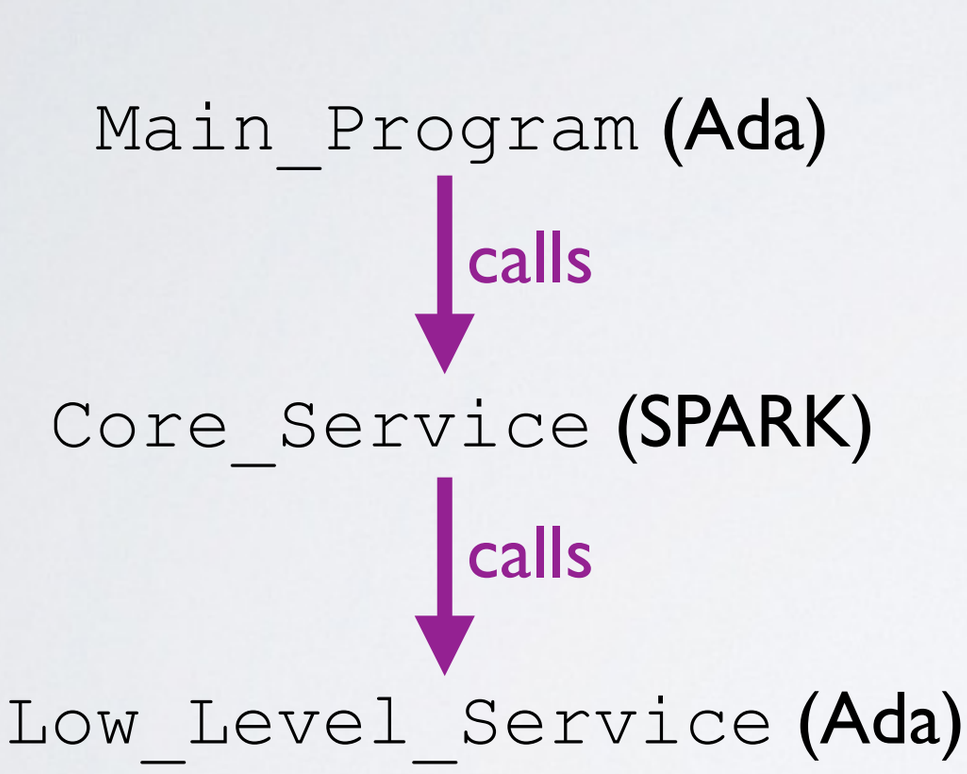
need to prove while-loop termination? use Loop_Variant

```
function Init return T_Variables is
   Result : T_Variables;
begin
   pragma Assert (Variable.Is_Valid (Var => Variable.C_Variable));
   for Variable_Id in T_Variable_Id loop
      Result (Variable_Id) := Variable.C_Variable;
      pragma Loop_Invariant
        (for all I in T_Variable_Id range T_Variable_Id'First .. Variable_Id =>
           Variable.Is_Valid (Result (I)));
   end loop;
   return Result;
end Init;
```

Combining Test & Proof

goal: be at least as good as test alone

strategy presented in *Integrating formal program verification with testing*, ERTS 2012
& *Explicit assumptions - a prenup for marrying static and dynamic program verification*, Test & Proof 2014



verification method

integration tests

formal verification

units tests

assumptions

Pre respected
non-aliasing of inputs
initialization of inputs

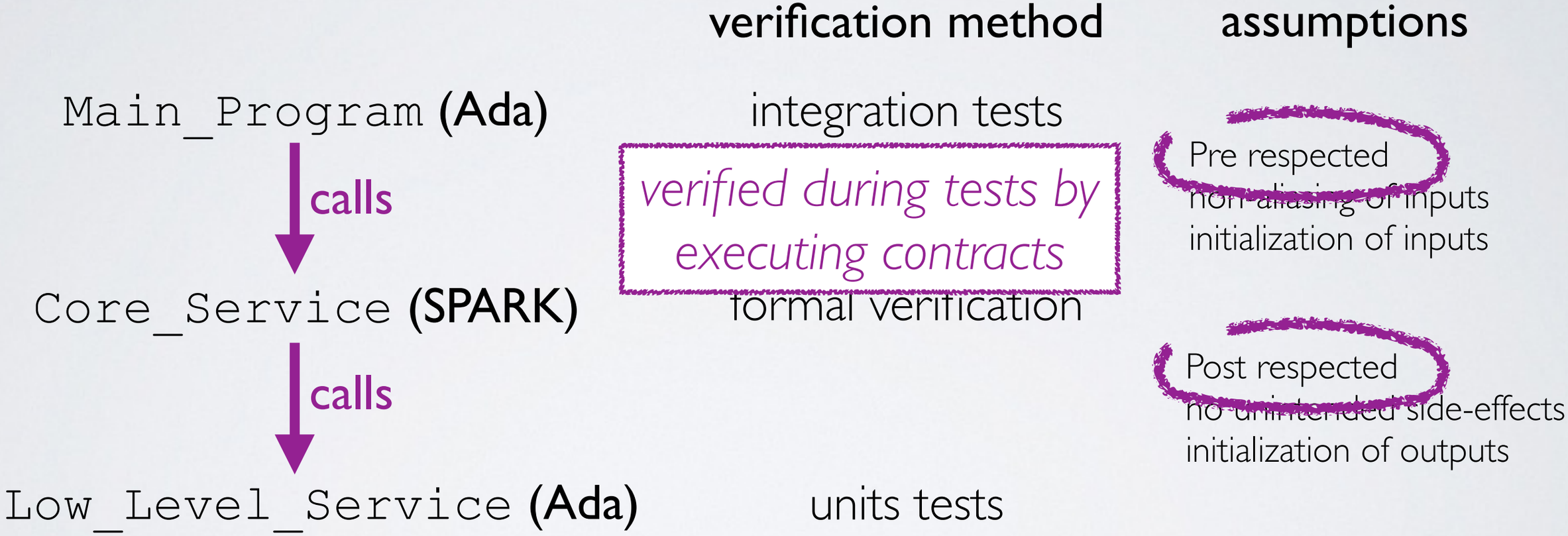
Post respected
no unintended side-effects
initialization of outputs

Combining Test & Proof

goal: be at least as good as test alone

strategy presented in *Integrating formal program verification with testing*, ERTS 2012

& *Explicit assumptions - a prenup for marrying static and dynamic program verification*, Test & Proof 2014

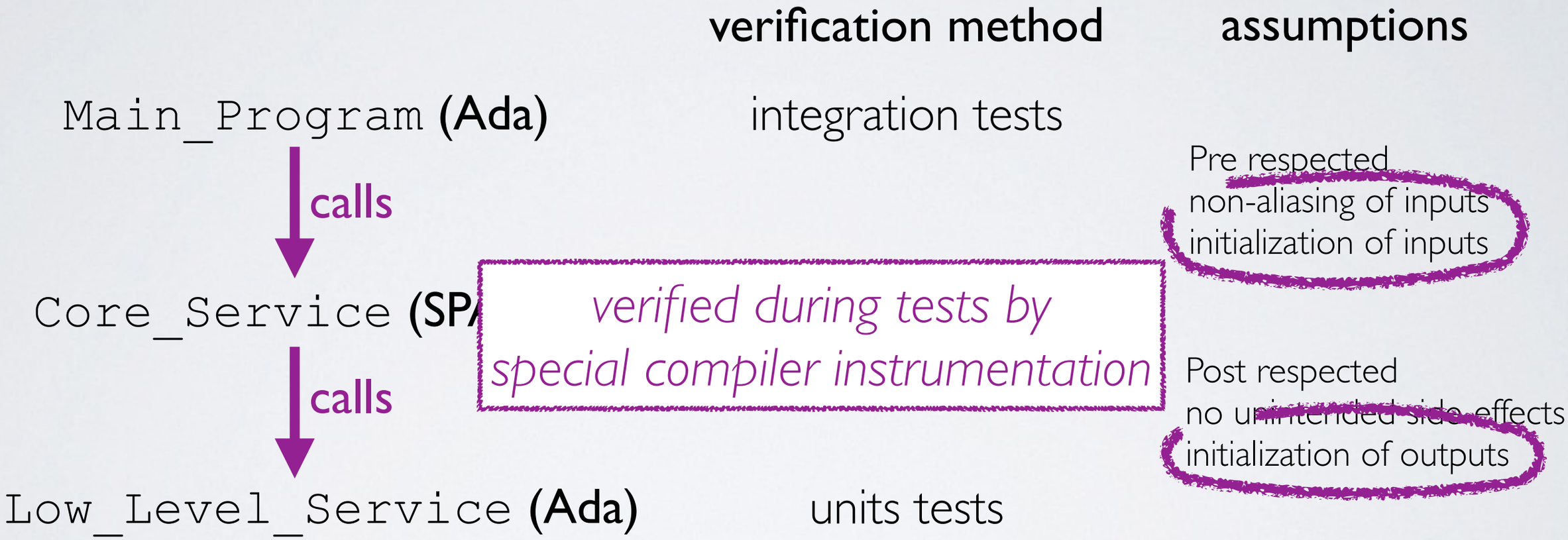


Combining Test & Proof

goal: be at least as good as test alone

strategy presented in *Integrating formal program verification with testing*, ERTS 2012

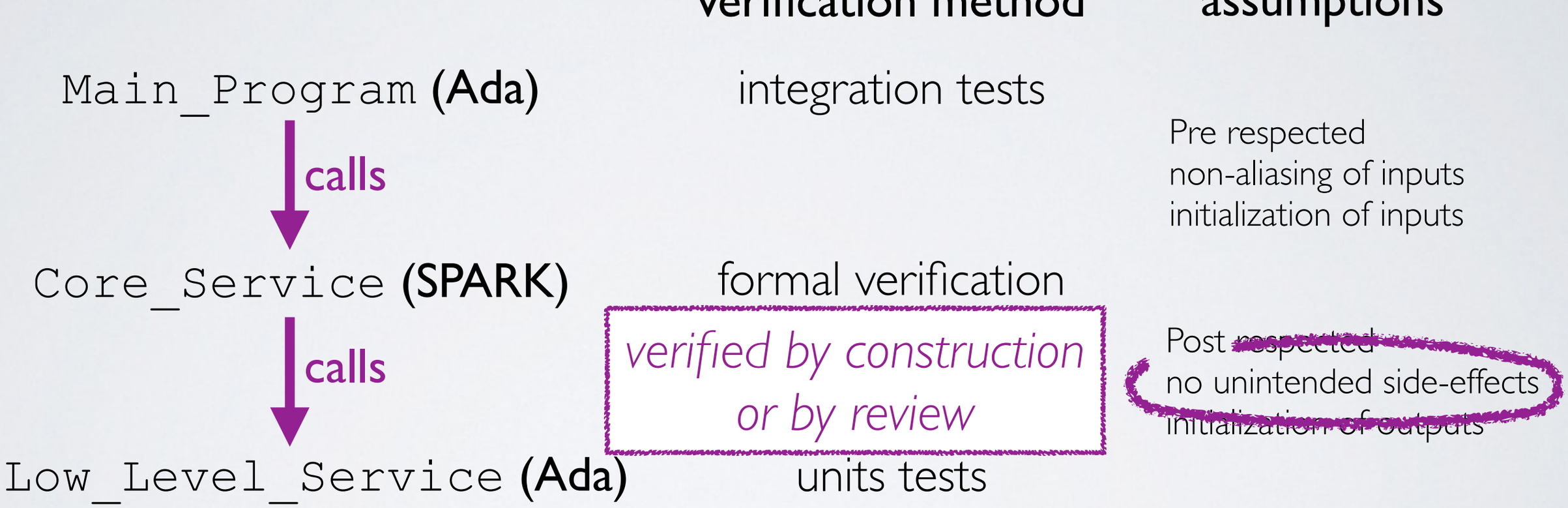
& *Explicit assumptions - a prenup for marrying static and dynamic program verification*, Test & Proof 2014



Combining Test & Proof

goal: be at least as good as test alone

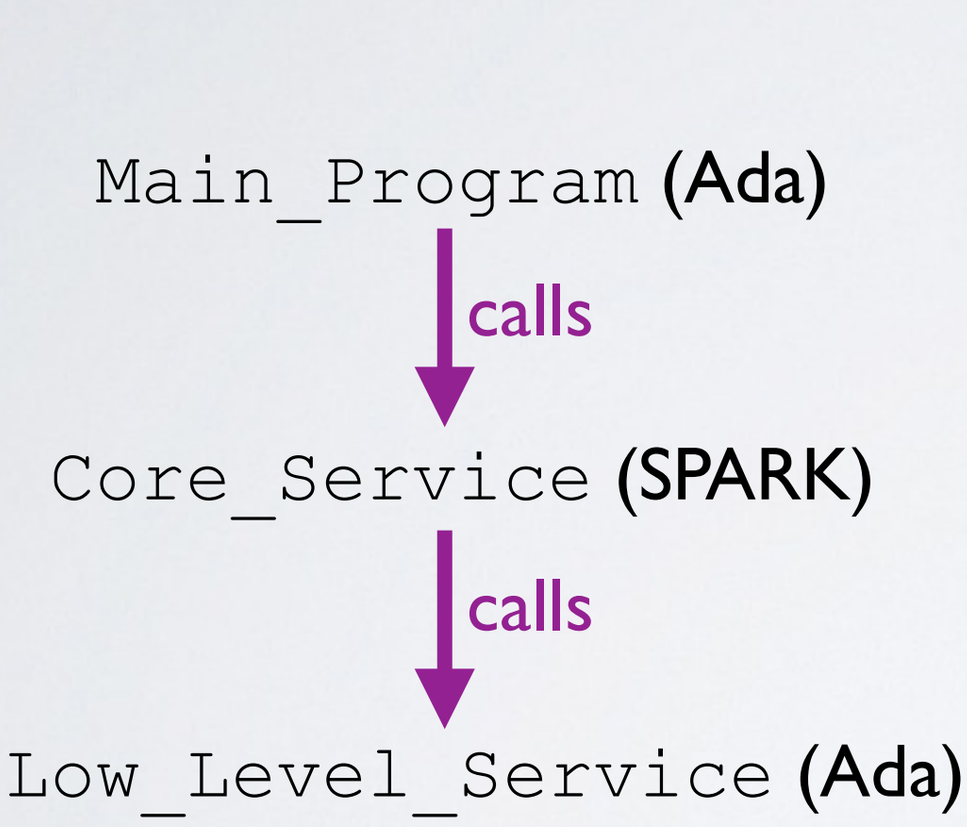
strategy presented in *Integrating formal program verification with testing*, ERTS 2012
& *Explicit assumptions - a prenup for marrying static and dynamic program verification*, Test & Proof 2014



Combining Test & Proof

goal: be at least as good as test alone

strategy presented in *Integrating formal program verification with testing*, ERTS 2012
& *Explicit assumptions - a prenup for marrying static and dynamic program verification*, Test & Proof 2014



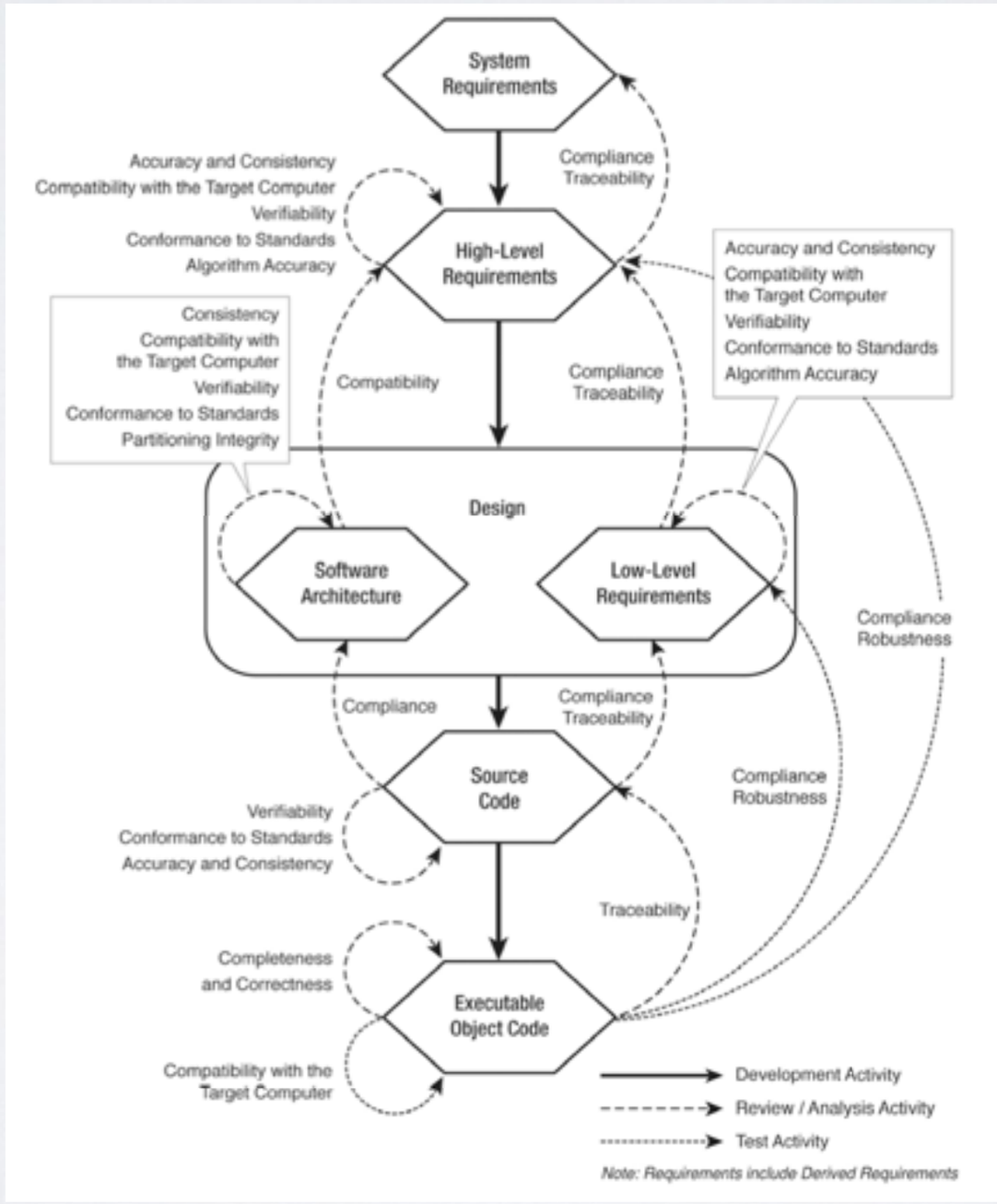
known objective of formal verification projects: justify assumptions

proposal: switch from ad-hoc to tool-assisted assumptions management

Test & Proof in DO-178C

goal: be at least as good as test alone, for all objectives assigned to test

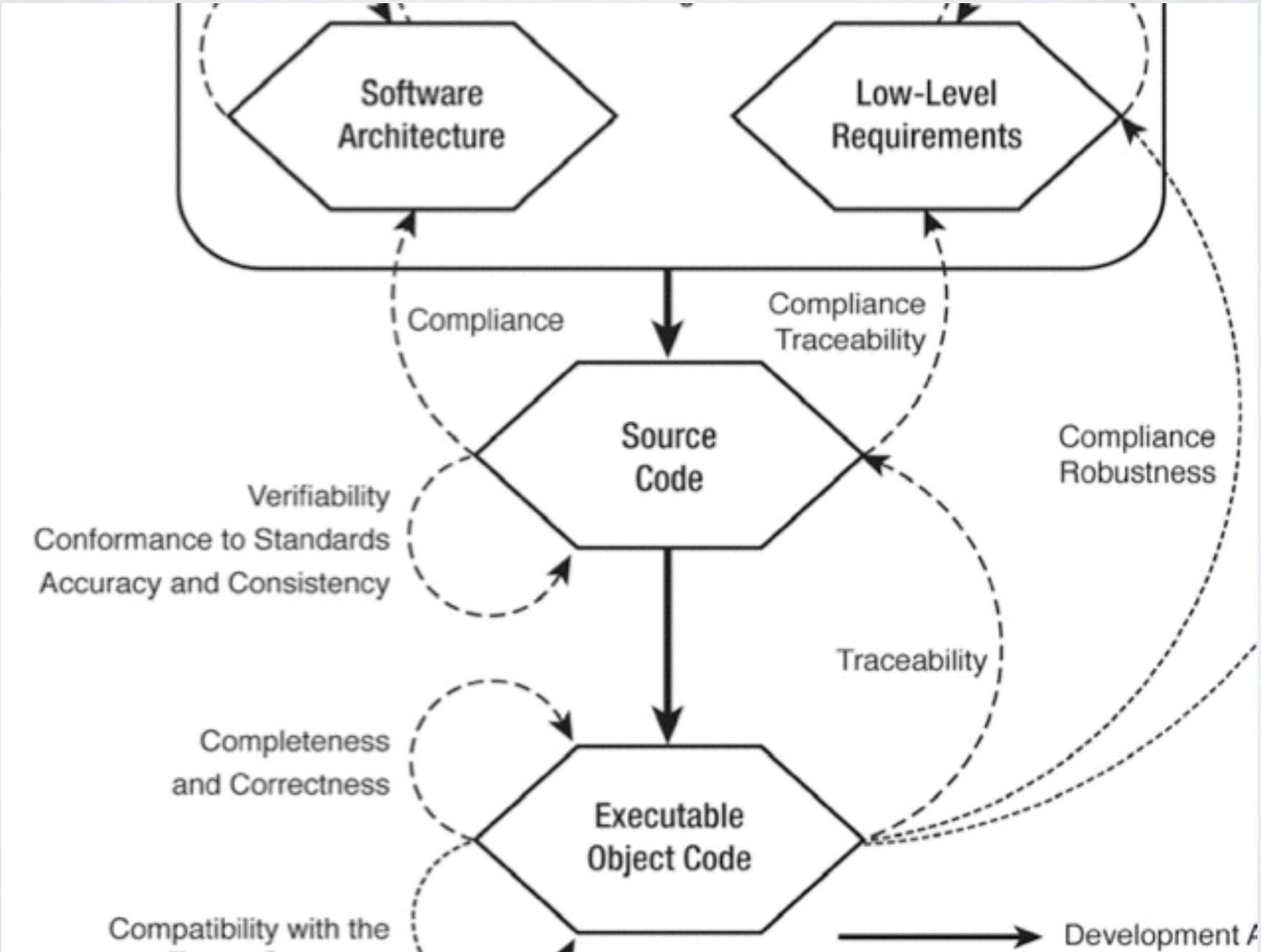
Testing or Formal Verification: DO-178C Alternatives and Industrial Experience, IEEE Software, June 2013
& *Guidelines for the Use of Theorem Proving in the Certification of Critical Systems*, workshop TPC, 2014



Test & Proof in DO-178C

goal: be at least as good as test alone, for all objectives assigned to test

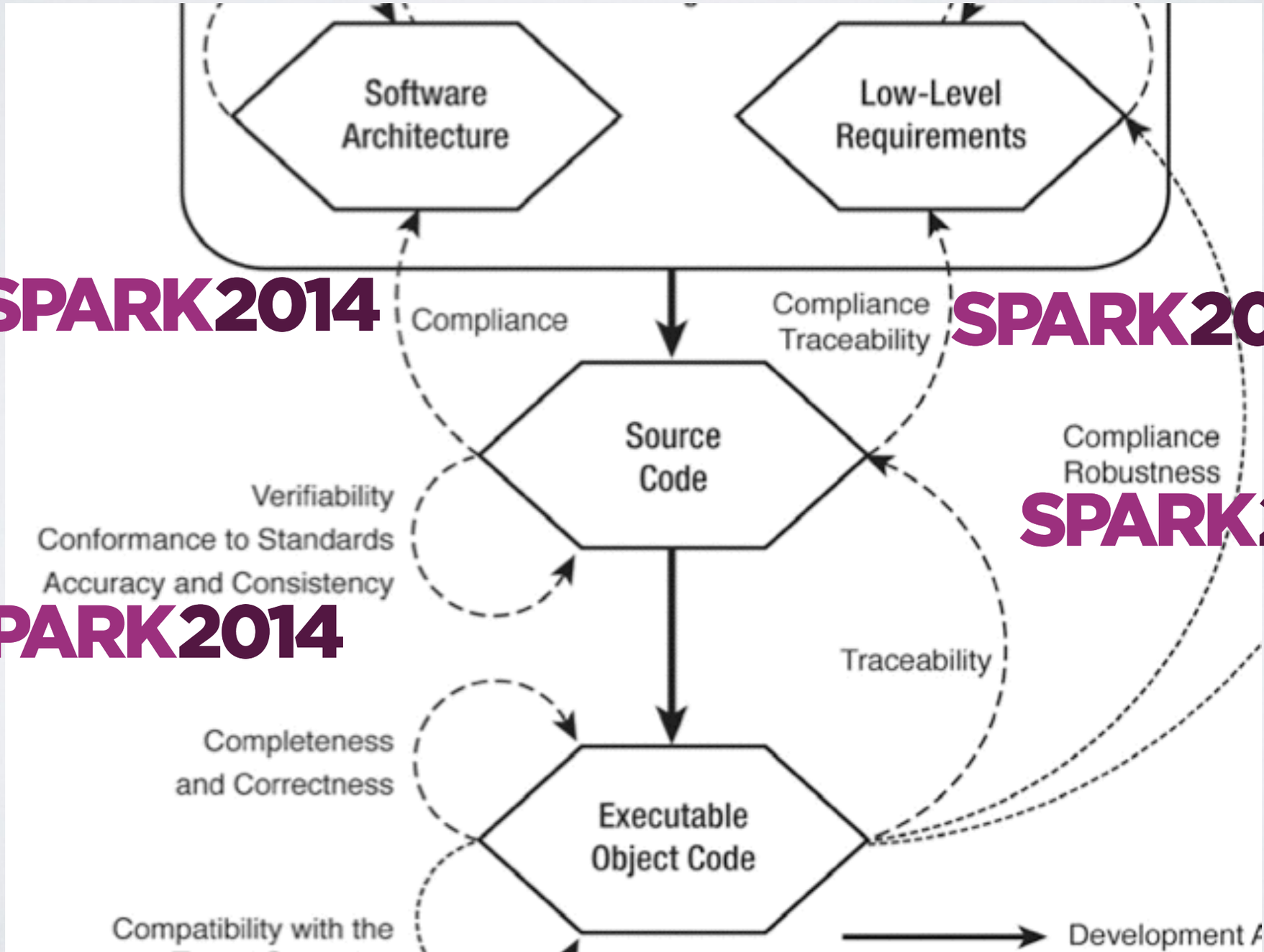
Testing or Formal Verification: DO-178C Alternatives and Industrial Experience, IEEE Software, June 2013
& *Guidelines for the Use of Theorem Proving in the Certification of Critical Systems*, workshop TPC, 2014



Test & Proof in DO-178C

goal: be at least as good as test alone, for all objectives assigned to test

Testing or Formal Verification: DO-178C Alternatives and Industrial Experience, IEEE Software, June 2013
& *Guidelines for the Use of Theorem Proving in the Certification of Critical Systems*, workshop TPC, 2014



SPARK2014

SPARK2014

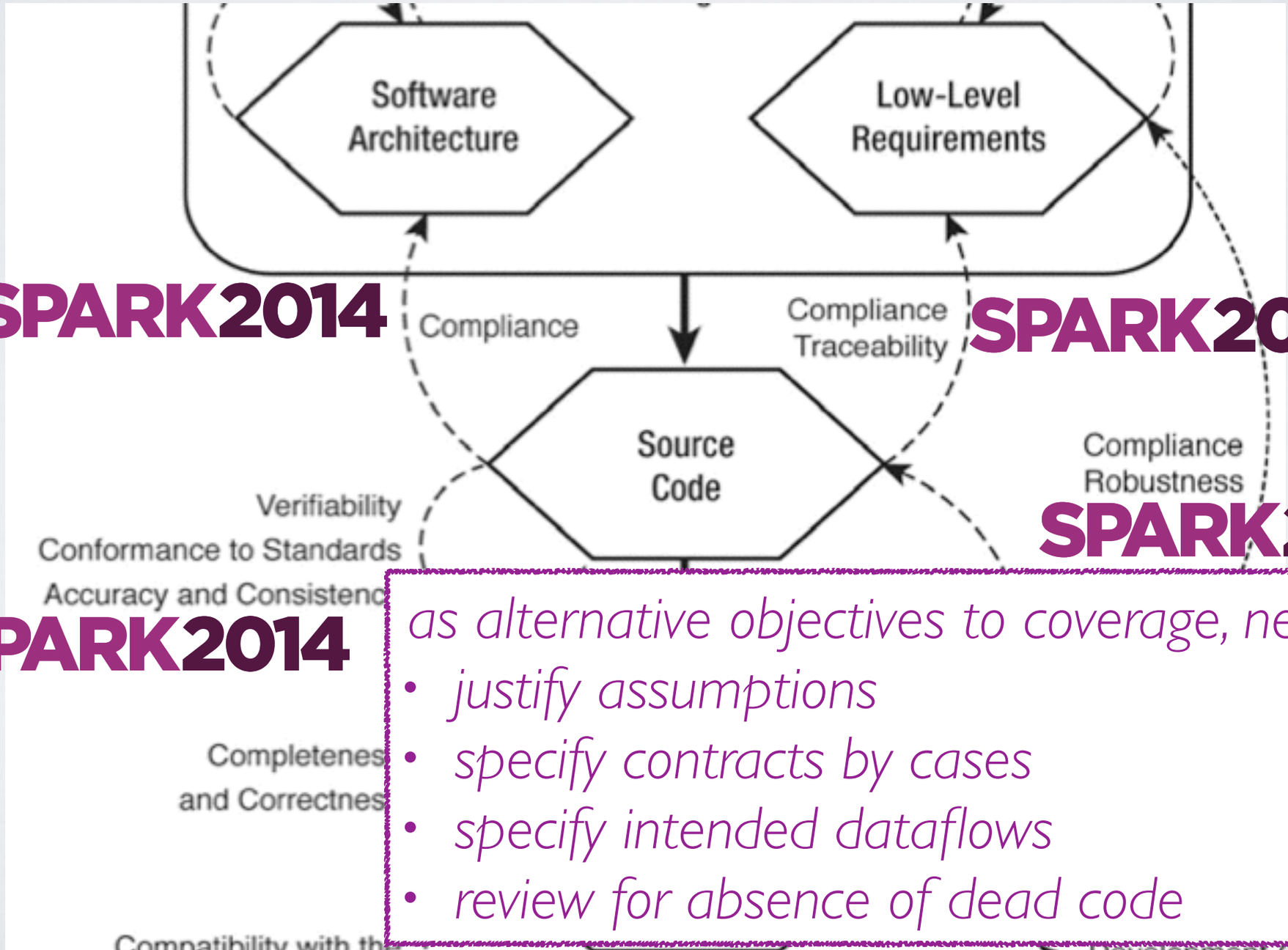
SPARK2014

SPARK2014

Test & Proof in DO-178C

goal: be at least as good as test alone, for all objectives assigned to test

Testing or Formal Verification: DO-178C Alternatives and Industrial Experience, IEEE Software, June 2013
& *Guidelines for the Use of Theorem Proving in the Certification of Critical Systems*, workshop TPC, 2014



SPARK2014

SPARK2014

SPARK2014

SPARK2014

as alternative objectives to coverage, need to:

- *justify assumptions*
- *specify contracts by cases*
- *specify intended dataflows*
- *review for absence of dead code*

SPARK2014 is the only language and toolset providing industrial support for both dynamic and formal contract-based verification of software.

http://www.adacore.com/sparkpro

http://www.spark-2014.org